BMC Bioinformatics

BioMed Central

Methodology article

Open Access

Selecting additional tag SNPs for tolerating missing data in genotyping

Yao-Ting Huang[1], Kui Zhang[3], Ting Chen[4] and Kun-Mao Chao*[1,2]

Address: [1]Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, [2]Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan, [3]Section on Statistical Genetics, Department of Biostatistics, University of Alabama at Birmingham, USA and [4]Department of Biological Sciences, University of Southern California, Los Angeles, CA 90089, USA

Email: Yao-Ting Huang - ythuang@acb.csie.ntu.edu.tw; Kui Zhang - kzhang@ms.soph.uab.edu; Ting Chen - tingchen@usc.edu; Kun-Mao Chao* - kmchao@csie.ntu.edu.tw

* Corresponding author

Published: 01 November 2005

BMC Bioinformatics 2005, **6**:263 doi:10.1186/1471-2105-6-263

Received: 26 May 2005

Accepted: 01 November 2005

This article is available from: http://www.biomedcentral.com/1471-2105/6/263

Abstract

Background: Recent studies have shown that the patterns of linkage disequilibrium observed in human populations have a block-like structure, and a small subset of SNPs (called tag SNPs) is sufficient to distinguish each pair of haplotype patterns in the block. In reality, some tag SNPs may be missing, and we may fail to distinguish two distinct haplotypes due to the ambiguity caused by missing data.

Results: We show there exists a subset of SNPs (referred to as robust tag SNPs) which can still distinguish all distinct haplotypes even when some SNPs are missing. The problem of finding minimum robust tag SNPs is shown to be NP-hard. To find robust tag SNPs efficiently, we propose two greedy algorithms and one linear programming relaxation algorithm. The experimental results indicate that (1) the solutions found by these algorithms are quite close to the optimal solution; (2) the genotyping cost saved by using tag SNPs can be as high as 80%; and (3) genotyping additional tag SNPs for tolerating missing data is still cost-effective.

Conclusion: Genotyping robust tag SNPs is more practical than just genotyping the minimum tag SNPs if we can not avoid the occurrence of missing data. Our theoretical analysis and experimental results show that the performance of our algorithms is not only efficient but the solution found is also close to the optimal solution.

Background

In recent years, *Single Nucleotide Polymorphisms* (SNPs) have become the preferred marker for association studies of genetic diseases or traits. A set of linked SNPs on one chromosome is called a *haplotype*. Recent studies have shown that the patterns of *Linkage Disequilibrium* (LD) observed in human populations have a block-like structure [4,13]. The chromosome recombination only takes place at some low LD regions called recombination hotspots. The high LD region between these hotspots is often referred to as a "haplotype block." Within a haplotype block, there is little or even no recombination occurred, and the SNPs in the block tend to be inherited together. Due to the low haplotype diversity within a block, the information carried by these SNPs is highly redundant. Thus, a small subset of SNPs (called "tag SNPs") is sufficient to distinguish each pair of patterns in the block [7,13,17-19]. Haplotype blocks with corre-

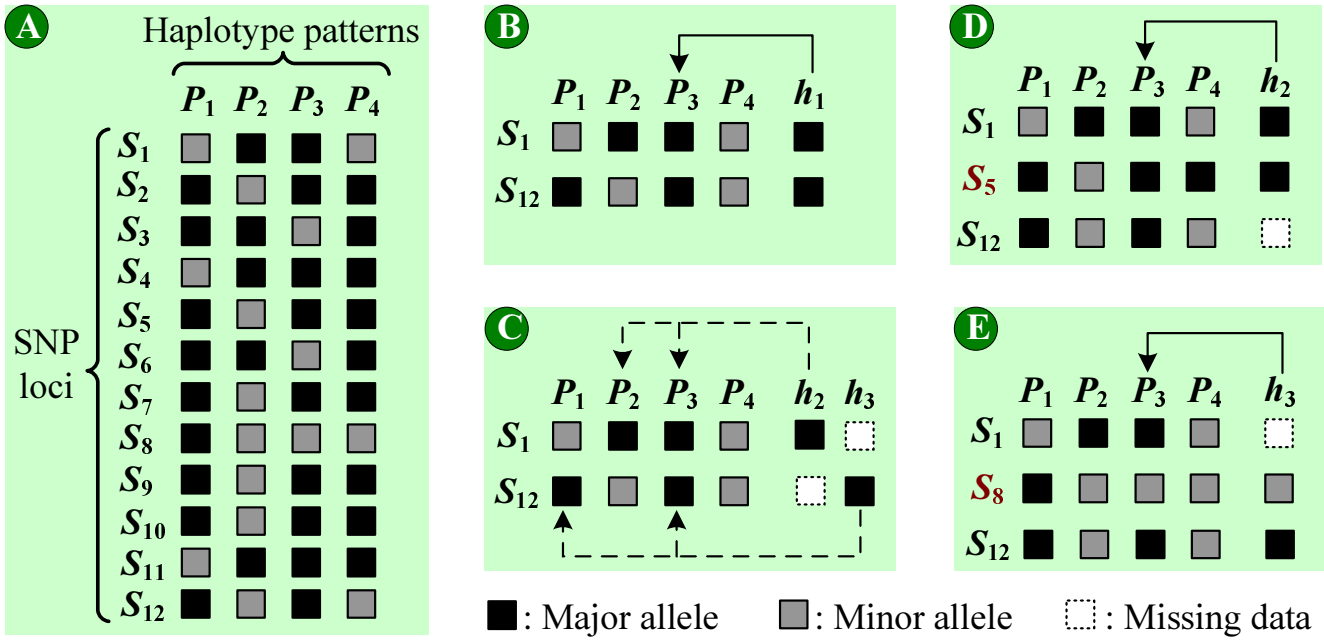

Figure 1

The influence of missing data and auxiliary tag SNPs. (A) A haplotype block defined by 12 SNPs and 4 haplotype patterns. Each column represents a haplotype pattern and each row represents a SNP locus. The black and grey boxes stand for the major and minor alleles at each SNP locus, respectively. (B) Tag SNPs genotyped without missing data. (C) Tag SNPs genotyped with missing data. (D) The auxiliary tag SNP S_5 for h_2. (E) The auxiliary tag SNP S_8 for h_3.

sponding tag SNPs are quite useful and cost-effective for association studies as it does not require genotyping all SNPs. Many studies have tried to find the minimum set of tag SNPs in a haplotype block. In a large-scale study of human Chromosome 21, Patil *et al.* [13] developed a greedy algorithm to partition the haplotypes into 4,135 blocks with 4,563 tag SNPs. Zhang *et al.* [17-19] used a dynamic programming approach to reduce the numbers of blocks and tag SNPs to 2,575 and 3,562, respectively. Bafna *et al.* [1] showed that the problem of minimizing tag SNPs is NP-hard and gave efficient algorithms for special cases of this problem.

In reality, a SNP may not be genotyped and considered to be missing data (i.e., we fail to obtain the allele configuration of the SNP) if it does not pass the threshold of data quality [13,16,19,20]. These missing data may cause ambiguity when using the minimum set of tag SNPs to distinguish an unknown haplotype sample. Figure 1 illustrates the influence of missing data when identifying haplotype samples. In this figure, a haplotype block (see Figure 1 (A)) defined by 12 SNPs and 4 haplotype patterns is presented (from the public haplotype data of human Chromosome 21 [13]). We follow the same assumption as previous studies that all SNPs are diallelic (i.e., taking on only two values) [1,13]. Suppose we select SNPs S_1 and S_{12} as tag SNPs. The haplotype sample h_1 is identified as haplotype pattern P_3 unambiguously (see Figure 1 (B)). Consider haplotype samples h_2 and h_3 with one missing tag SNP (see Figure 1 (C)). h_2 can be identified as haplotype patterns P_2 or P_3, and h_3 can be identified as P_1 or P_3. As a result, these missing tag SNPs result in ambiguity when distinguishing unknown haplotype samples.

Although we can not avoid the occurrence of missing data, the remaining SNPs within the haplotype block may provide abundant information to resolve the ambiguity. For example, if we re-genotype an additional SNP S_5 for h_2 (see Figure 1 (D)), h_2 is identified as haplotype pattern P_3 unambiguously. On the other hand, if SNP S_8 is re-genotyped (see Figure 1 (E)), h_3 is also identified unambiguously. These additional SNPs are referred to as "auxiliary tag SNPs," which can be found from the remaining SNPs in the block and are able to resolve the ambiguity caused by missing data.

Alternatively, instead of re-genotyping auxiliary tag SNPs whenever encountering missing data, we work on a set of SNPs which is not affected by the occurrence of missing data. Figure 2 illustrates a set of SNPs which can tolerate one missing SNP. Suppose we select SNPs S_1, S_5, S_8, and S_{12} to be genotyped. Note that no matter which SNP is missing, each of the 16 missing patterns can be distinguished by the remaining three SNPs. Therefore, all hap-

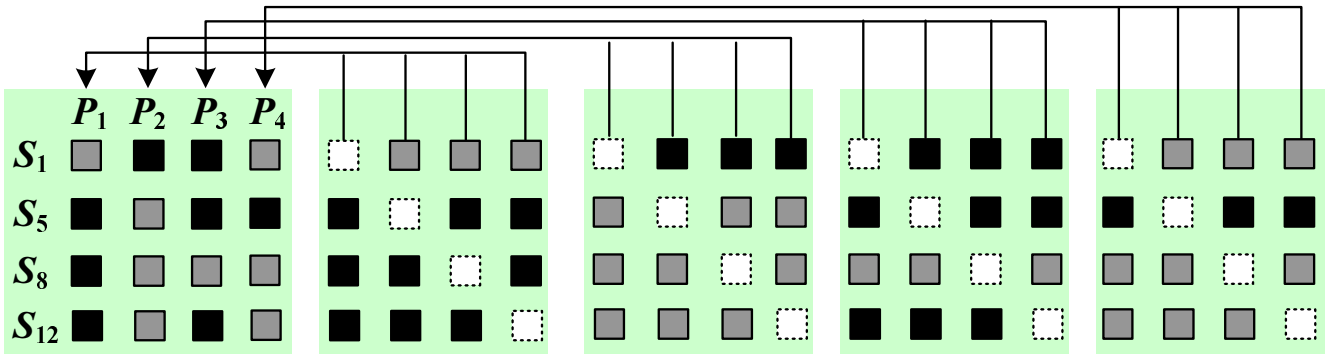

Figure 2
The robust tag SNPs. A set of robust tag SNPs for tolerating one missing tag SNP.

lotype samples with one missing SNP can still be identified unambiguously. We refer to these SNPs as "robust tag SNPs," which are able to tolerate a number of missing data. The important feature of robust tag SNPs is that although they consume more SNPs than the "tag SNPs" defined in previous studies, they guarantee that all haplotype samples with a number of missing data can be distinguished unambiguously. When the occurrence of missing data is frequent, the cost of re-genotyping processes can be reduced by robust tag SNPs.

This paper focuses on the problem of finding robust tag SNPs to tolerate a number of missing data. Throughout this paper, we denote m as the maximum number of missing SNPs to be tolerated, which corresponds to different missing rates in different genotyping experiments. And we wish to find a minimum set of robust tag SNPs which can distinguish each pair of haplotypes even when up to m SNPs are missing. We assume that the haplotype phases and block partition are available as the input. Numerous methods have been developed to infer haplotypes from genotype data [12,14,15]. Several algorithms have also been proposed to find the block partition [4,13,17]. The problem of finding minimum robust tag SNPs is shown to be NP-hard (See Theorem 1). To find robust tag SNPs efficiently, we propose two greedy algorithms and one linear programming (LP) relaxation algorithm. The proposed algorithms have been implemented and tested on a variety of simulated and empirical data. We also analyze the efficiency and solutions of these algorithms. An algorithm for finding auxiliary tag SNPs is described assuming robust tag SNPs have been computed in advance.

Results

We propose two greedy algorithms which select the robust tag SNPs one by one in different greedy manners. In addition, we reformulate this problem as an integer programming problem and design an LP-relaxation algorithm to solve this problem. The greedy and LP-relaxation algorithms are able to find solutions within factors of $(m + 1)\ln\frac{K(K-1)}{2}$, $\ln((m+1)\frac{K(K-1)}{2})$, and $O(m \ln K)$ of the optimal solution respectively, where m is the maximum number of missing SNPs allowed and K is the number of haplotype patterns in the block.

We have implemented the first and second greedy algorithms in JAVA [see Additional files 1 and 2]. The LP-relaxation algorithm has been implemented in Perl [see Additional file 3], where the LP problem is solved via a program called "lp_solve" [11]. The LP-relaxation algorithm is a randomized method. Thus, this program is repeated for 10 times to explore different solutions and the best solution among them is chosen as the output.

In order to evaluate the solutions and efficiency of our algorithms, we also implement a program in JAVA (referred to as "OPT") which uses a brute force method to find the optimal solution. For a given data set of N SNPs, the OPT program examines all possible solutions (i.e., all subsets of $\binom{N}{1}$, $\binom{N}{2}$, ..., and $\binom{N}{N}$). The minimum subset of SNPs that can tolerate m missing SNPs is chosen as the output. Due to the NP-hardness of this problem, the OPT program fails to output the optimal solution within a reasonable period of time in many data sets. As a consequence, we skip some impossible solution space to speed up this program by the following two observations: (1) the solutions with less than or equal to m SNPs are the impossible ones since m SNPs might be missing; and (2) for a data set containing K haplotype patterns, the minimum number of SNPs required to distinguish each of them is at least log K (see Lemma 2). As a result, we can

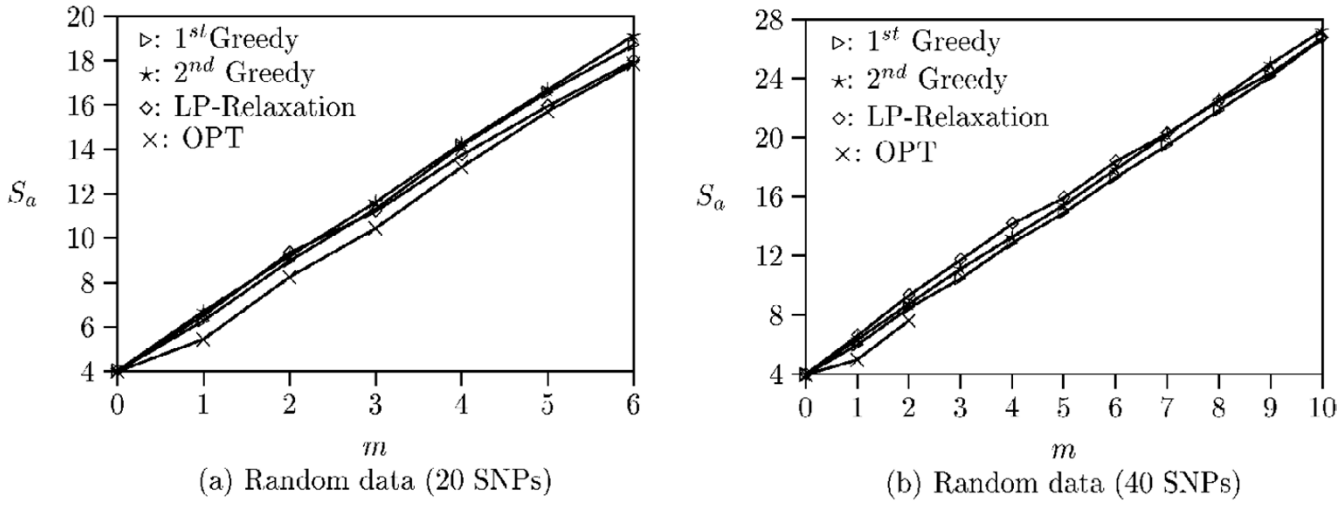

Figure 3
Experimental results on random data. (a) Results from data sets containing 10 haplotypes and 20 SNPs. (b) Results from data sets containing 10 haplotypes and 40 SNPs.

examine the possible solutions only for subsets of $\binom{N}{m+\log K}$, $\binom{N}{m+\log K+1}$ ..., and $\binom{N}{N}$. By searching possible solutions from small subsets to large ones, the OPT program can stop and output the optimal solution immediately when a subset that can tolerate m missing SNPs is found.

Results on simulated data

Theoretically, all SNPs will reach complete linkage equilibrium after sufficient chromosome recombination takes place. We first generate 100 data sets containing short haplotypes which simulate this bottleneck model [12,14,15]. Each data set consists of 10 haplotypes with 20 SNPs. These haplotypes are created by randomly assigning the major or minor alleles at each SNP locus. Let

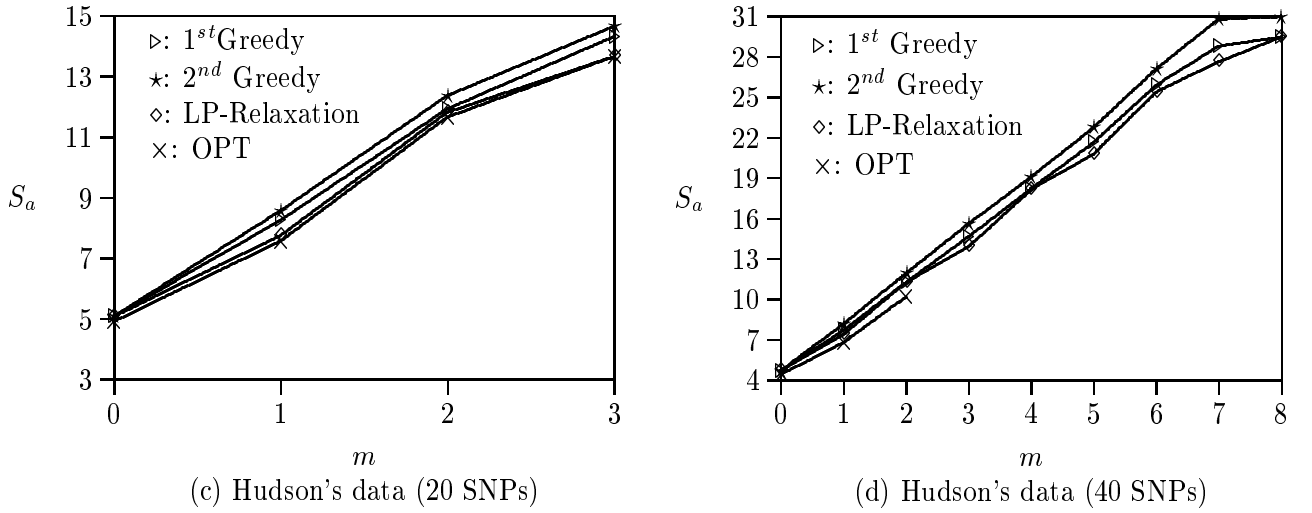

Figure 4
Experimental results on Hudson's data. (a) Results from data sets containing 10 haplotypes and 20 SNPs. (b) Results from data sets containing 10 haplotypes and 40 SNPs.

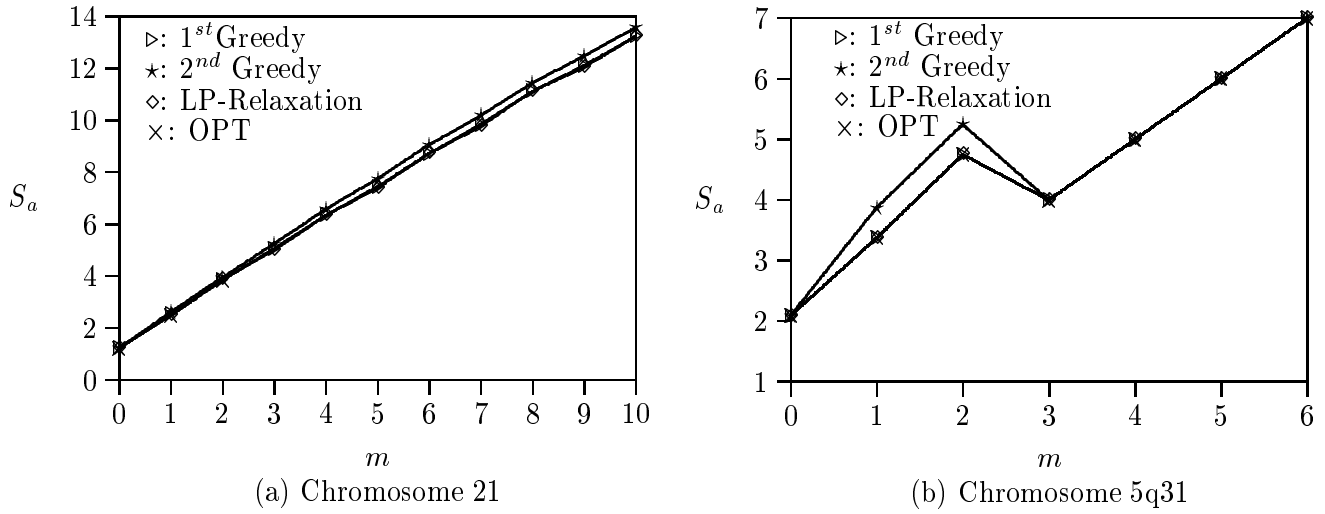

Figure 5
Experimental results on real data. (a) Results from Patil's Chromosome 21 data, (b) Results from Daly's Chromosome 5q31 data.

m be the number of missing SNPs allowed and S_a be the average number of robust tag SNPs over 100 data sets. Figure 3 (a) plots S_a with respect to m (roughly corresponding to SNP missing rates from 0% to 33%). When m = 0, all programs find the same number of SNPs as the optimal solution. The iterative LP-relaxation algorithm slightly outperforms the others as m increases. When m > 6, more than 20 SNPs are required to tolerate missing data. Thus, no data sets contain enough SNPs for solutions.

We then generate 100 data sets containing long haplotypes. Each data set is composed of 10 haplotypes with 40 SNPs. Figure 3 (b) illustrates the experimental results on these long data sets (corresponding to SNP missing rates from 0% to 37%). The optimal solutions for $m > 2$ can not be found by the OPT program within a reasonable period of time (after one week computation) and are not shown in this figure. It is because the possible solutions in long data sets are too large to enumerate. On the other hand, both greedy and iterative LP-relaxation algorithms run in polynomial time and always output a solution efficiently. In this experiment, both greedy algorithms slightly outperforms the iterative LP-relaxation algorithm. In addition, the number of missing SNPs allowed is larger than those in short data sets. For example, to tolerate 10 missing SNPs (i.e., $m = 10$), all programs output less than 28 SNPs. The remaining SNPs in each data set are still sufficient to tolerate more missing SNPs.

Hudson (2002) [10] provides a program which can simulate a set of haplotypes under the assumption of neutral evolution and uniformly distributed recombination rate using the coalescent model. We use Hudson's program to generate 100 short data sets with 10 haplotypes and 20 SNPs and 100 long data sets with 10 haplotypes and 40 SNPs. Figure 4 (a) shows the experimental results on Hudson's short data sets (corresponding to SNP missing rates from 0% to 23%). The number of missing SNPs allowed are less than that of random data. It is because Hudson's program generates coalescent haplotypes which are similar to each other. As a result, many SNPs can not be used to distinguish haplotypes and the amount of tag SNPs is inadequate to tolerate larger missing SNPs. In this experiment, we observe that the iterative LP-relaxation algorithm finds solutions quite close to the optimal solutions and slightly outperforms the other two algorithms.

Figure 4 (b) illustrates the experimental results on long data sets generated by Hudson's program (corresponding to SNP missing rates from 0% to 29%). The optimal solutions for $m > 2$ again can not be found by the OPT program within a reasonable period of time. In this experiment, the performance of the first greedy and iterative LP-relaxation algorithms are similar, and they slightly outperform the second greedy algorithm as m becomes large.

Results on real data

We also test these programs on two real data sets: (1) public haplotype data of human Chromosome 21 released by Patil et al. [13]; and (2) a 500 KB region on human Chromosome 5q31 which may contain a genetic variant related to the Crohn disease by Daly et al. [4]. Patil's data include 20 haplotypes of 24,047 SNPs spanning over

Table 1: The detailed result of first greedy algorithm on Daly's 11 blocks.

Block ID	1	2	3	4	5	6	7	8	9	10	11	S_a
$m = 0$	1	1	2	3	3	2	3	2	2	2	2	23/11 = 2.09
$m = 1$	2	2	*f*	5	*f*	3	5	4	*f*	3	3	27/8 = 3.375
$m = 2$	3	3	*f*	8	*f*	*f*	*f*	*f*	*f*	5	*f*	19/4 = 4.75
$m = 3$	4	4	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	8/2 = 4
$m = 4$	5	5	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	10/2 = 5
$m = 5$	6	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	6/1 = 6
$m = 6$	7	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	*f*	7/1 = 7

f: fail to contain enough SNPs for tolerating *m* missing SNPs

about 32.4 MB, which are partitioned into 4,135 haplotype blocks. By genotyping 103 SNPs with minor allele frequency at least 5%, Daly et al. partition the 500 KB region into 11 haplotype blocks. Each haplotype block in these real data sets contains different numbers of SNPs and haplotypes (e.g., from several SNPs to hundreds of SNPs). When m increases, some short blocks may not contain enough SNPs for tolerating missing data (e.g., m > the number of SNPs in a block). As a consequence, S_a here stands for the average number of robust tag SNPs over those blocks still containing solutions.

Figure 5 (a) shows the experimental results on Patil's 4,135 blocks. Because there are many long blocks in Patil's data (e.g., more than one hundred SNPs), the optimal solution for $m > 2$ can not be found within a reasonable period of time. The experimental result indicates that all algorithms find similar number of robust tag SNPs when *m* is small. The LP-relaxation algorithm slightly outperforms the others as *m* increases.

Figure 5 (b) illustrates the experimental results on Daly's 11 blocks. Because the haplotype blocks partitioned by Daly *et al*. are very short (e.g., most blocks contain less than 12 SNPs), all optimal solutions still can be found. The solutions found by each algorithm is almost the same as optimal solutions. Theoretically, S_a should grow monotonically as *m* increases. But due to the small number of blocks in Daly's data set, S_a does not grow smoothly when *m* increases from 2 to 3. To explain this phenomenon, we report the detailed result of the first greedy algorithm in Table 1. For each of the 11 blocks, the number of robust tag SNPs found with respect to different values of *m* is listed in the table. Note that as mentioned before, some blocks may not contain enough SNPs for tolerating large missing data as *m* increases. When *m* increases from 2 to 3, Blocks 4 and 10 (which consumes 8 and 5 SNPs) do not contain enough SNPs for a solution and are discarded. As a result, S_a (for $m = 3$) is computed only using Blocks 1 and 2 and the value is lower than the previous one (i.e., from 4.75 to 4). This phenomenon is not shown in Figure 5 (a) because it is amortized by thousands of blocks in Patil's data set.

Discussion

In terms of efficiency, the first and second greedy algorithms are faster than the LP-relaxation algorithm. The greedy algorithms usually returns a solution in seconds and the LP-relaxation algorithm requires about half minute for a solution. It is because the running time of LP-relaxation algorithm is bounded by the time of solving the LP problem. Furthermore, this LP-relaxation algorithm is repeated for 10 times to explore 10 different solutions. The OPT program for searching the optimal solution is apparently slower than the others. The optimal solution usually can not be found within a reasonable period of time if the size of the block becomes large. ¿From our empirical study, the optimal solution can be found in reasonable time by the OPT program if the block contains less than 20 SNPs (e.g., the short random data sets). But

Table 2: The number of total tag SNPs found by each algorithm. The percentage of tag SNPs with respect to total SNPs is shown in parentheses.

	Random data		Hudson's data		Patil's data	Daly's data
Total blocks	100	100	100	100	4135	11
Total SNPs	2000	4000	2000	4000	24047	103
1st Greedy	400 (20%)	400 (10%)	509 (25.5%)	472 (11.8%)	4610 (19.2%)	23 (22.3%)
2nd Greedy	400 (20%)	400 (10%)	509 (25.5%)	472 (11.8%)	4610 (19.2%)	23 (22.3%)
LP-relaxation	400 (20%)	400 (10%)	509 (25.5%)	471 (11.8%)	4657 (19.4%)	23 (22.3%)
OPT	400 (20%)	400 (10%)	492 (24.6%)	443 (11.1%)	4595 (19.1%)	23 (22.3%)

Table 3: The tradeoffs between additional tag SNPs required and maximum missing rates allowed. These results come from the first greedy algorithm applied on random and Hudson's data sets with 40 SNPs.

	m	0	1	2	3	4	5
Random data (40 SNPs)	average number of robust tag SNPs	4	6	8.51	10.47	12.89	14.92
	corresponding SNP missing rate	0	16.7%	23.5%	28.6%	31.0%	33.5%
	average number of extra tag SNPs	0	2	4.51	6.47	8.89	10.92
Hudson's data (40 SNPs)	average number of robust tag SNPs	4.72	7.71	11.28	14.67	18.23	21.67
	corresponding SNP missing rate	0	13.0%	17.7%	20.4%	21.9%	23.1%
	average number of extra tag SNPs	0	2.99	6.56	9.95	13.51	16.95

for those large data sets with more than 40 SNPs, the OPT program is significantly outperformed by the approximation algorithms (e.g., fail to output a solution within one week computation).

Assuming no missing data (i.e., $m = 0$), we compare the solutions found by each algorithm with the optimal solution. Table 2 lists the numbers of total tag SNPs found by each algorithm in previous experiments. In the experiments on random and Daly's data, the solution found by each algorithm is as good as the optimal solution. In the experiments on Hudson's and Patil's data, these algorithms still find solutions quite close to the optimal solution. For example, the approximation ratios of these algorithms are only $\frac{472}{443} \approx 1.07$ and $\frac{4657}{4595} \approx 1.01$, respectively.

We then analyze the genotyping cost that can be saved by using tag SNPs. In Table 2, the percentage of tag SNPs in each data set is shown in parentheses. The experimental results indicate that the cost of genotyping tag SNPs is significantly reduced in comparison with genotyping all SNPs in a block. For example, in Patil's data, we only need to genotype about 19% of tag SNPs in each block, which saves about 81% genotyping cost.

The tradeoffs between the number of additional tag SNPs required and the number of missing SNPs allowed are discussed in the following. In practice, missing data in the genotyping experiment are usually limited to certain missing rate. We transform the maximum number of missing SNPs allowed into maximum missing rates allowed by calculating the percentage of m with respect to the number of robust tag SNPs. Table 3 lists the results of the first greedy algorithm applied on random and Hudson's long data sets. The number of additional tag SNPs grows with respect to m linearly. However, we observe that the maximum missing rate allowed grows slowly as m becomes large. This is because more additional tag SNPs are required in order to tolerate more missing SNPs. But under the same SNP missing rate, genotyping these additional tag SNPs may also increase the number of missing SNPs, which reduces the power of robust tag SNPs. On the positive side, when m is small, the corresponding maximum missing rate allowed is sufficient for most genotyping experiments since their missing rates are usually less than 10%. For example, the robust tag SNPs with $m = 1$ are sufficient to tolerate 10% missing SNPs, and they only requires at most 3 additional SNPs. As a result, genotyping additional tag SNPs for tolerating missing data is cost-effective under the current genotyping environment.

In reality, not all haplotypes are of equal importance or confidence. When selecting robust tag SNPs, it might be desirable to weight them according to their population frequency. To incorporate the frequency of haplotypes into this problem, there are two possible ways:

1. It can be easily done by discarding the rare haplotypes and retain the common haplotypes as the input of our algorithms. This approach would not require modification to our algorithms. But the retained common haplotypes will be processed as equally weighted.

2. Our algorithms try to find a set of SNPs such that each pair of haplotypes are distinguished by a threshold of at least $(m + 1)$ SNPs. A simplest way to weight the haplotypes is choosing different thresholds for each pair of haplotypes according to their population frequency. The haplotype pairs with higher frequency can then be assigned with more tag SNPs than the lower ones by our algorithms.

Conclusion

In this paper, we show there exists a set of robust tag SNPs which is able to tolerate a number of missing data. Our study indicates that genotyping robust tag SNPs is more practical than genotyping minimum tag SNPs for association studies if we can not avoid the occurrence of missing data. We describe two greedy and one LP-relaxation

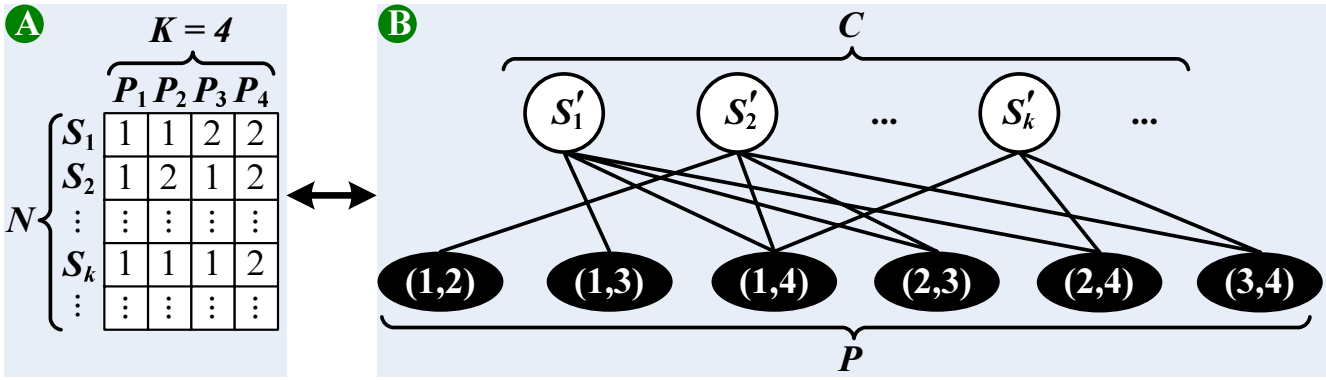

Figure 6
Reformulation of the MRTS problem. (A) The haplotype matrix M_h containing N SNPs and K haplotype patterns. (B) The bipartite graph corresponding to M_h.

approximation algorithms for finding robust tag SNPs. Our experimental results and theoretical analysis show that these algorithms are not only efficient but the solutions found are also close to the optimal solution. In terms of genotyping cost, we observe that the genotyping cost saved by using robust tag SNPs is significant, and genotyping additional tag SNPs to tolerate missing data is still cost-effective. One future direction is to assign weights to different types of SNPs (e.g., SNPs in coding or non-coding regions), and design algorithms for the selection of weighted tag SNPs.

Software availability

Project name: efficient algorithms for utilizing SNP information.

Project home page: http://www.csie.ntu.edu.tw/~kmchao/tools/Robust_Tag_SNP

Operating system: the implemented greedy algorithms are platform independent, and the implemented LP-relaxation algorithm runs on the Windows operating system.

Programming language: the greedy algorithms are implemented in JAVA, and the LP-relaxation algorithm is implemented in Perl.

Methods

Assume we are given a haplotype block containing N SNPs and K haplotype patterns. This block is denoted by an $N \times K$ binary matrix M_h (see Figure 6 (A)). Define $M_h[i,j] \in \{1,2\}$ for each $i \in [1, N]$ and $j \in [1, K]$, where 1 and 2 represent the major and minor alleles, respectively. In reality, the haplotype block may also contain missing data. This formulation can be easily extended to handle missing data by treating them as wild card symbols. To simplify the presentation of this paper, we will assume no missing data in the block. Let C be the set of all SNPs in M_h. The robust tag SNPs $C' \subseteq C$ are a subset of SNPs which is able to distinguish each pair of haplotype patterns unambiguously when at most m SNPs are missing. Note that the missing data may occur at any SNP locus and thus create different missing patterns (see Figure 2). For any haplotype pattern with up to m missing SNPs, the set of robust tag SNPs C' is required to distinguish all of them unambiguously.

To distinguish a haplotype pattern unambiguously, each pair of patterns must be distinguished by at least one SNP in C'. For example (see Figure 6 (A)), we say patterns P_1 and P_2 can be distinguished by SNP S_2 since $M_h[2,1] \neq M_h[2,2]$. A formal definition of this problem is given below.

Problem: Minimum Robust Tag SNPs (MRTS)
Input: An N × K matrix M_h and an integer m.

Output: The minimum subset of SNPs $C' \subseteq C$ which satisfies:

(1) for each pair of patterns P_i and P_j, these is a SNP $S_k \in C'$ such that $M_h[k, i] \neq M_h[k, j]$;

(2) when at most m SNPs are discarded from C' arbitrarily, (1) still holds.

We then reformulate MRTS to a variant of the *set covering problem* [6]. Each SNP $S_k \in C$ (i.e., the k-th row in M_h) is reformulated to a set $S'_k = \{(i, j) \mid M[k, i] \neq M[k, j]$ and $i < j\}$. For example, suppose the k-th row in M_h is $\{1,1,1,2\}$. The corresponding set $S'_k = \{(1,4), (2,4), (3,4)\}$. In other

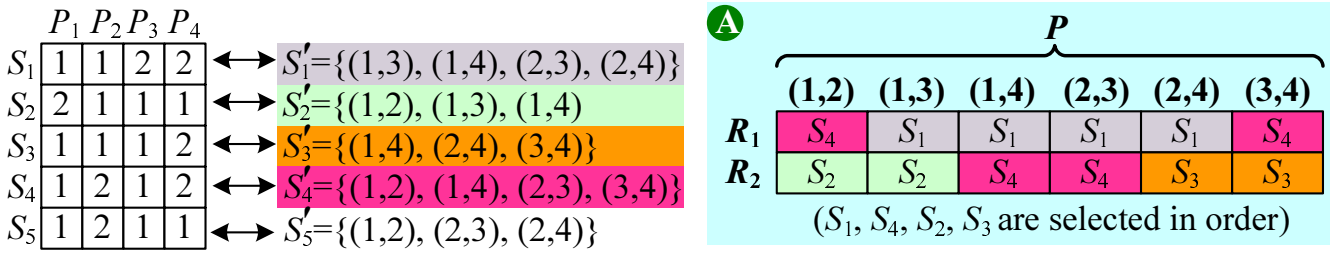

Figure 7
An example of the first greedy algorithm. The SNPs S_1, S_4, S_2, and S_3 are selected by the first greedy algorithm. (A) The table that stores each selected SNP.

words, S_k' stores the pairs of patterns distinguished by SNP S_k. Define P as the set that contains all pairs of patterns (i.e., $P = \{(i,j) \mid 1 \le i < j \le K\} = \{(1,2), (1,3), ..., (K-1,K)\}$).

Consider each element in P and each reformulated set of C as nodes in an undirected bipartite graph (see Figure 6 (B). If SNP S_k can distinguish patterns P_i and P_j (i.e., $(i,j) \in S_k'$), there is an edge connecting the nodes (i, j) and S_k'. The following lemma implies that each pair of patterns must be distinguished by at least $(m + 1)$ SNPs to tolerate m missing SNPs.

Lemma 1. *$C' \subseteq C$ is the set of robust tag SNPs which allows at most m missing SNPs iff each node in P has at least $(m + 1)$ edges connecting to each node in C'.*

Proof. Let C' be the set of robust tag SNPs which allows at most m missing SNPs. Suppose patterns P_i and P_j are distinguished by only m SNPs in C' (i.e., (i, j) has only m edges connecting to nodes in C'). However, if these m SNPs are all missing, no other SNPs in C' are able to distinguish patterns P_i and P_j, which is a contradiction. Thus, each pair of patterns must be distinguished by at least $(m + 1)$ SNPs, which implies that each node in P must have at least $(m + 1)$ edges connecting to nodes in C'. The proof of the other direction is similar.

In the following, we give a lower bound regarding the minimum number of robust tag SNPs required, which is used to skip some solution space by the OPT program.

Lemma 2. *Given K haplotype patterns, the minimum number of robust tag SNPs required is at least* $\log K$.

Proof. Recall that the value of a SNP is binary. The maximum number of distinct haplotypes which can be distinguished by N SNPs is at most 2^N. As a result, for a given data set containing K haplotype patterns, the minimum number of SNPs required is at least $\log K$.

The following theorem shows the NP-hardness of the MRTS problem, which implies there is no polynomial time algorithm to find the optimal solution of MRTS.

Theorem 1. *The MRTS problem is NP-hard.*

Proof. When $m = 0$, MRTS is the same as the original problem of finding minimum tag SNPs, which is known as the *minimum test set* problem [6,17]. Since the minimum test set problem is NP-hard and can be reduced to a special case of MRTS, MRTS is NP-hard.

The first greedy algorithm

To solve MRTS efficiently, we propose a greedy algorithm which returns a solution not too larger than the optimal solution. By Lemma 1, to tolerate m missing tag SNPs, we need to find a subset of SNPs $C' \subseteq C$ such that each pair of patterns in P is distinguished by at least (m + 1) SNPs in C'. Assume that the SNPs selected by this algorithm are stored in a (m + 1) × |P| table (see Figure 7 (A)). Initially, each grid in the table is empty. Once a SNP S_k, (that can distinguish patterns P_i and P_j) is selected, one grid of the column (i, j) is filled in with S_k, and we say that this grid is covered by S_k.

This greedy algorithm works by covering the grids from the first row to the $(m + 1)$-th row, and greedily selects a SNP which covers most uncovered grids in the i-th row at each iteration. In other words, while working on the i-th row, a SNP is selected if its reformulated set S' maximizes $|S' \cap R_i|$, where R_i is the set of uncovered grids at the i-th row.

Figure 7 illustrates an example for this algorithm to tolerate one missing tag SNP (i.e., $m = 1$). The SNPs S_1, S_4, S_2, and S_3 are selected in order. When all grids in this table are covered, each pair of patterns is distinguished by $(m + 1)$ SNPs in the corresponding column. Thus, the SNPs in this

table are the robust tag SNPs which can tolerate up to m missing SNPs. The pseudo code of this greedy algorithm is given below.

Algorithm: FIRST-GREEDY-ALGORITHM (C, P, m)

1 $R_i \leftarrow P$, $\forall i \in [1, m + 1]$

2 $C' \leftarrow \phi$

3 **for** i = 1 to m + 1 **do**

4 **while** $R_i \neq \phi$ **do**

5 select and remove a SNP S from C that maximizes $|S' \cap R_i|$

6 $C' \leftarrow C' \cup S$

7 $j \leftarrow i$

8 **while** $S' \neq \phi$ **and** $j \leq m + 1$ **do**

9 $S_{tmp} \leftarrow S' \cap R_j$ //S_{tmp} is a temporary variable for holding the result of $S' \cap R_i$

10 $R_j \leftarrow R_j - S_{tmp}$

11 $S' \leftarrow S' - S_{tmp}$

12 $j \leftarrow j + 1$

13 **endwhile**

14 **endwhile**

15 **endfor**

16 **return** C'

The time complexity of this algorithm is analyzed as follows. At Line 4, the number of iterations of the intermediate loop is bounded by $|R_i| \leq |P|$. Within the loop body (Lines 5–13), Line 5 takes $O(|C||P|)$ because we need to check all SNPs in C and examine the uncovered grids of R_i. The inner loop (Lines 8–13) takes only $O(|S'|)$. Thus, the entire program runs in $O(m|C||P|^2)$.

We now show the solution C' returned by the first greedy algorithm is not too larger than the optimal solution C^*. Suppose the algorithm selects the k-th SNP when working on the i-th row. Let $|S_k^c|$ be the number of grids in the i-th row covered by the k-th selected SNP (i.e., $|S_k^c| = |S' \cap R_i|$; see Line 5 in FIRST-GREEDY-ALGORITHM). For example (see Figure 7), $S_2^c = 2$ since the second selected SNP (i.e., S_4) covers two grids in the first row. We incur 1 unit of cost to each selected SNP, and spread this cost among the grids in S_k^c[3]. In other words, each grid at the i-th row and j-th column is assigned a cost C_j^i (see Figure 8), where

$$C_j^i = \begin{cases} \frac{1}{|S_k^c|} & \text{if the algorithm selects the } k\text{-th SNP when covering the } i\text{-th row;} \\ 0 & \text{otherwise.} \end{cases}$$

Since each selected SNP is assigned 1 unit of cost, the sum of C_j^i for each grid in the table is equal to $|C'|$,

i.e.,

$$|C'| = \sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i. \qquad (1)$$

Let R_k^i be the number of uncovered grids in the i-th row before the k-th iteration (i.e., (k - 1) SNPs have been selected by the algorithm). For example (see Figure 8), $R_2^1 = 2$ since two grids in the first row are still uncovered before the second SNP is selected. Define $C_i^{'}$ as the set of iterations used by the algorithm when working on the i-th row. For example (see Figure 8), $C_2^{'} = \{3,4\}$ since this algorithm works on the second row in the third and fourth iterations. We can rewrite (1) as

$$\sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i = \sum_{i=1}^{m+1} \sum_{k \in C_i^{'}} \left(R_{k-1}^i - R_k^i \right) \frac{1}{|S_k^c|}. \qquad (2)$$

Lemma 3. *The k-th selected SNP has* $|S_k^c| \geq \frac{R_{k-1}^i}{|C^*|}$.

Proof. Suppose the algorithm is working on the i-th row at the beginning of the k-th iteration. Let C_k^* be the set of SNPs in C^* (the optimal solution) that has been selected by the algorithm before the k-th iteration, and the set of

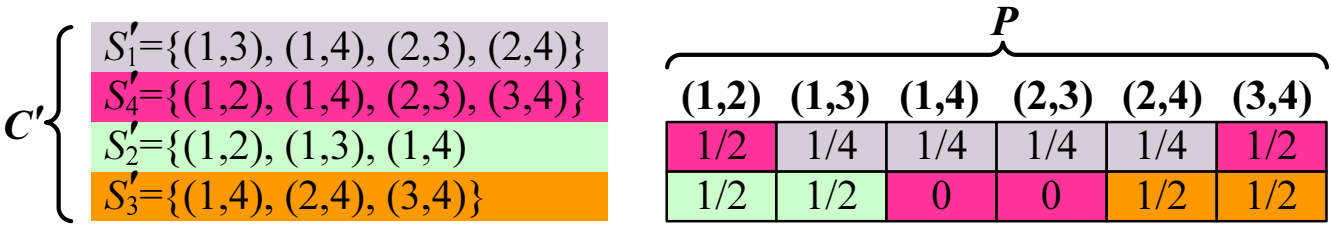

Figure 8

Analysis of the first greedy algorithm. This figure shows the cost C_j^i of each grid for the first greedy algorithm.

remaining SNPs in C^* be $C_{\bar{k}}^*$. We claim that there exists a SNP in $C_{\bar{k}}^*$ which can cover at least $\frac{R_k^i}{|C_{\bar{k}}^*|}$ grids in the *i*-th row. Otherwise (i.e., each SNP in $C_{\bar{k}}^*$ covers less than $\frac{R_k^i}{|C_{\bar{k}}^*|}$ grids), all SNPs in $C_{\bar{k}}^*$ will cover less than $(\frac{R_k^i}{|C_{\bar{k}}^*|} \times |C_{\bar{k}}^*| = R_k^i)$ grids in the *i-th* row. But since $C_k^* \bigcup C_{\bar{k}}^* = C^*$, this implies that C^* can not cover all grids in R_k^i, which is a contradiction. Because all SNPs in $C_{\bar{k}}^*$ are candidates to the greedy algorithm, the *k*-th selected SNP must cover at least $\frac{R_k^i}{|C_{\bar{k}}^*|}$ grids in the *i*-th row, which implies $|S_k^c| \geq \frac{R_{k-1}^i}{|C^*|}$ since $|C^*| \geq |C_{\bar{k}}^*|$ and $|R_k^i| \leq |R_{k-1}^i|$. □

Theorem 2. *The first greedy algorithm gives a solution of* $(m + 1) \ln \frac{K(K-1)}{2}$ *approximation.*

Proof. Define the *d*-th harmonic number as $H(d) = \sum_{i=1}^{d} \frac{1}{i}$ and $H(0) = 0$. By (2) and Lemma 3,

$$\begin{aligned}
\sum_{i=1}^{m+1} \sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i &= \sum_{i=1}^{m+1} \sum_{k \in C_i} \left(R_{k-1}^i - R_k^i\right) \frac{1}{|S_k^c|} \leq \sum_{i=1}^{m+1} \sum_{k \in C_i} \left(R_{k-1}^i - R_k^i\right) \frac{|C^*|}{R_{k-1}^i} \\
&= \sum_{i=1}^{m+1} \sum_{k \in C_i} \left(\sum_{l=R_k^i+1}^{R_{k-1}^i} \frac{|C^*|}{R_{k-1}^i} \right) \\
&\leq |C^*| \sum_{i=1}^{m+1} \sum_{k \in C_i} \sum_{l=R_k^i+1}^{R_{k-1}^i} \frac{1}{l} \qquad \left(l \leq R_{k-1}^i\right) \\
&= |C^*| \sum_{i=1}^{m+1} \sum_{k \in C_i} \left(\sum_{l=1}^{R_{k-1}^i} \frac{1}{l} - \sum_{l=1}^{R_k^i} \frac{1}{l} \right) \\
&\leq |C^*| \sum_{i=1}^{m+1} \sum_{k \in C_i} \left(H(R_{k-1}^i) - H(R_k^i)\right) \\
&\leq |C^*| \sum_{i=1}^{m+1} \left(H(R_0^i) - H(R_{|C_i|}^i)\right) \\
&\leq |C^*|(m+1) \max\{H(R_0^i)\} \qquad (R_{|C_i|}^i = 0 \text{ and } H(0) = 0) \\
&\leq |C^*|(m+1) \ln |P|. \qquad (H(R_0^i) \leq H(|P|)) \qquad (3)
\end{aligned}$$

By (1) and (3), we get

$$\frac{|C'|}{|C^*|} \leq (m+1) \ln |P| = (m+1) \ln \frac{K(K-1)}{2}.$$

The second greedy algorithm

This section describes the second greedy algorithm which returns a solution of better approximation than that of the first greedy algorithm. Let R_i be the set of uncovered grids at the i-th row. Unlike the row-by-row manner of the first greedy algorithm, this algorithm greedily selects a SNP that covers most uncovered grids in the table (i.e., its reformulated set S' maximizing $|S' \cap (R_1 \cup \ldots \cup R_{m+1})|$). Let T be the collection of R_i (i.e., T is the set of all uncovered grids in the table). If the grids in the i-th row are all covered (i.e., $R_i = \phi$), R_i is removed from T. This algorithm runs until T = ϕ (i.e., all grids in the table are covered).

Figure 9 illustrates an example for this algorithm with *m* set to 1. The SNPs S_1, S_2, S_4, and S_5 are selected in order. Since this algorithm runs until all grids are covered, the set of SNPs in this table is able to tolerate *m* missing tag SNPs. The pseudo code of this algorithm is given below.

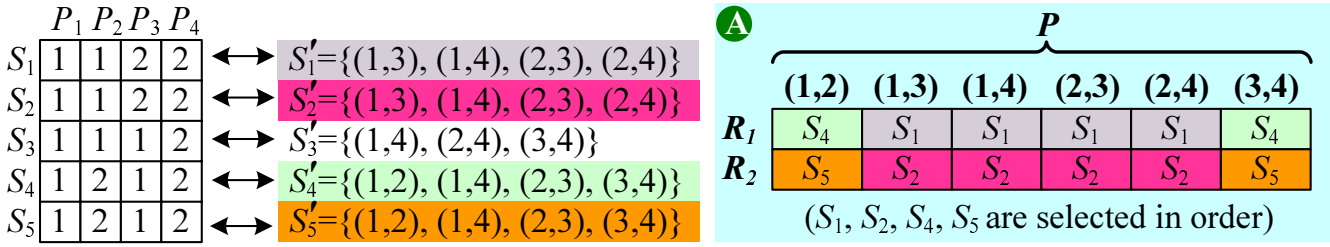

Figure 9
An example of the second greedy algorithm. The SNPs S_1, S_2, S_4, and S_5 are selected by the second greedy algorithm. (A) The table that stores each selected SNP.

Algorithm: SECOND-GREEDY-ALGORITHM (C, P, m)

1 $R_i \leftarrow P, \forall\ i \in [1, m+1]$

2 $T \leftarrow \{R_1, R_2, \ldots, R_{m+1}\}$

3 $C' \leftarrow \phi$

4 **while** $T \neq \phi$ **do**

5 select and remove a SNP S from C that maximizes $|S' \cap (R_1 \cup \ldots \cup R_{m+1})|$

6 $C' \leftarrow C' \cup S$

7 **for** each $R_i \in T$ **and** $S' \neq \emptyset$ **do**

8 $S_{tmp} \leftarrow S' \cap R_i$ // S_{tmp} is a temporary variable for holding the result of $S' \cap R_i$

9 $R_i \leftarrow R_i - S_{tmp}$

10 $S' \leftarrow S' - S_{tmp}$

11 **if** $R_i = \phi$ **then** $T \leftarrow T - R_i$

12 **endfor**

13 **end while**

14 **return** C'

The time complexity of this algorithm is analyzed as follows. At Line 4, the number of iterations of the loop is bounded by $O(|T|) = O(m|P|)$. Within the loop, Line 5 takes $O(|C||P|)$ time because we need to check each SNP in C and examine if it can cover any uncovered grid in each column. The inner loop (Lines 7–12) is bounded by $O(|S'|) < O(|P|)$. Thus, the running time of this program is $O(m|C||P|^2)$.

We now evaluate the solution returned by the second greedy algorithm. Let C' and C^* be the set of SNPs selected by this algorithm and the optimal solution, respectively. Let $|S_k^c|$ be the number of grids in the table covered by the k-th selected SNP. For example (see Figure 9), $|S_2^c| = 4$ since the second selected SNP (i.e., S_2) covers four grids in the table. Define T_k as the number of uncovered grids in the table before the k-th iteration. We have the following lemma similar to Lemma 3.

Lemma 4. *The k-th selected SNP has* $\left|S_k^c\right| \geq \frac{T_{k-1}}{|C^*|}$.

Proof. The proof is similar to that of Lemma 3. Let C_k^* be the set of remaining SNPs in C^* which has not been selected before the k-th iteration. We claim that there exists a SNP in C_k^* which can cover at least $\frac{T_k}{\left|C_k^*\right|}$ grids in the table. Otherwise, we can get the same contradiction (i.e., C^* fails to cover all grids) as in Lemma 3. Since $|C^*| \geq |C_k^*|$ and $T_{k-1} \leq T_k$, we have $\left|S_k^c\right| \geq \frac{T_{k-1}}{|C^*|}$. □

Theorem 3. *The second greedy algorithm gives a solution of* $\ln((m+1)\frac{K(K-1)}{2})$ *approximation.*

Proof. Each grid at the i-th row and j-th column is assigned a cost $C_j^i = \frac{1}{\left|S_k^c\right|}$ (see Figure 10) if it is covered by the k-th selected SNP. The sum of C_j^i for each grid is

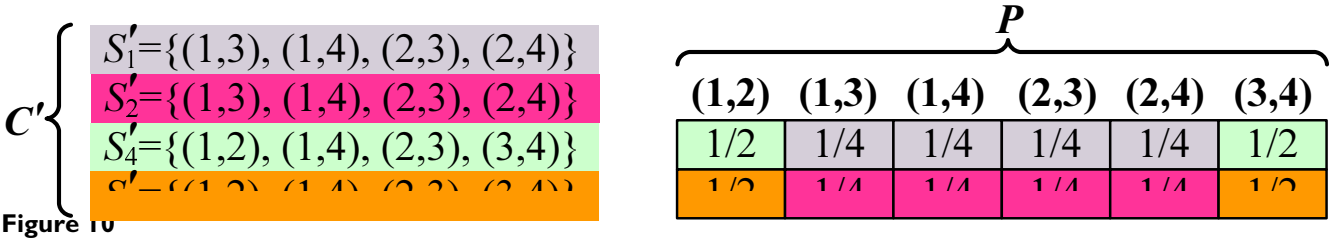

Figure 10

Analysis of the second greedy algorithm. This figure shows the cost C_j^i of each grid for the second greedy algorithm.

$$
\begin{aligned}
|C'| = \sum_{i=1}^{m+1}\sum_{j=1}^{\frac{K(K-1)}{2}} C_j^i &= \sum_{k=1}^{|C'|}(T_{k-1}-T_k)\frac{1}{\left|S_k^c\right|} && \text{(see (1) and (2))}\\
&\le \sum_{k=1}^{|C'|}(T_{k-1}-T_k)\frac{|C^*|}{T_{K-1}} && \text{(by Lemma 4)}\\
&\le |C^*|\,(H(T_0)-H(T_{|C'|})) && \text{(see the proof in Theorem 2)}\\
&\le |C^*|\ln((m+1)|P|). && (4)
\end{aligned}
$$

By (4), we have

$$\frac{|C'|}{|C^*|} \le \ln((m+1)|P|) = \ln((m+1)\frac{K(K-1)}{2}).$$

The iterative LP-relaxation algorithm

In practice, a probabilistic approach is sometimes more useful since the randomization can explore different solutions. In this section, we reformulate the MRTS problem to an Integer Programming (IP) problem. Based on the IP problem, we propose an iterative Linear Programming (LP)-relaxation algorithm. The iterative LP-relaxation algorithm is described below.

Step 1. Given a haplotype block containing N SNPs and K haplotype patterns. Let $\{x_1, x_2, ..., x_N\}$ be the set of integer variables for the N SNPs, where $x_k = 1$ if the SNP S_k is selected and $x_k = 0$ otherwise. Define $D(P_i, P_j)$ as the set of SNPs which are able to distinguish P_i and P_j patterns. By Lemma 1, to allow at most m missing SNPs, each pair of patterns must be distinguished by at least $(m + 1)$ SNPs. Therefore, for each set $D(P_i, P_j)$, at least $(m + 1)$ SNPs have to be selected to distinguish P_i and P_j patterns. As a consequence, the MRTS problem can be formulated as the following IP problem:

$$
\begin{aligned}
&\textbf{Minimize} && \sum_{k=1}^{N} x_k\\
&\textbf{Subject to} && \sum_{k\in D(P_i,P_j)} x_k \ge m+1, \qquad \text{for all } 1 \le i < j \le K, \qquad (5)\\
& && x_k = 0 \text{ or } 1.
\end{aligned}
$$

Step 2. Since solving the IP problem is NP-hard [6], we relax the integer constraint of x_k, and the IP problem becomes an LP problem defined as follows:

$$
\begin{aligned}
&\textbf{Minimize} && \sum_{k=1}^{N} y_k\\
&\textbf{Subject to} && \sum_{k\in D(P_i,P_j)} y_k \ge m+1, \qquad \text{for all } 1 \le i < j \le K, \qquad (6)\\
& && 0 \le y_k \le 1.
\end{aligned}
$$

The above LP problem can be solved in polynomial time by efficient algorithms such as the interior point method (Forsgren *et al.*, 2002) [5].

Step 3. Let $\{y_1, y_2, ..., y_N\}$ be the set of linear solutions obtained from (6), where $0 \le y_k \le 1$. We construct the corresponding integer solutions $\{x_1, x_2, ..., x_N\}$ by the following randomized rounding method:

$$
\text{Assign}\begin{cases} x_k = 1 \text{ with probability } y_k, \\ x_k = 0 \text{ with probability } 1\text{-}y_k. \end{cases}
$$

Note that the constructed integer solutions do not necessary satisfy all inequalities in (5). The randomized rounding method simply assigns x_k to 1 or 0 using the value of y_k as the likelihood, regardless of the inequalities in (5).

Step 4. We check whether the integer solutions constructed in Step 3 satisfy all inequalities in (5) or not.

Case 1. If some inequalities in (5) are still unsatisfied, we repeat Steps 1, 2, and 3 only for those unsatisfied inequalities until all of them are satisfied.

Case 2. If all inequalities in (5) are satisfied, we construct a final solution by setting $x_k = 1$ if x_k is assigned to 1 in any one of the iterations and setting $x_k = 0$ otherwise.

We now evaluate the solution returned by the iterative LP-relaxation algorithm. The selection of each SNP is considered as a *Bernoulli* random variable x_k taking values 1 (or

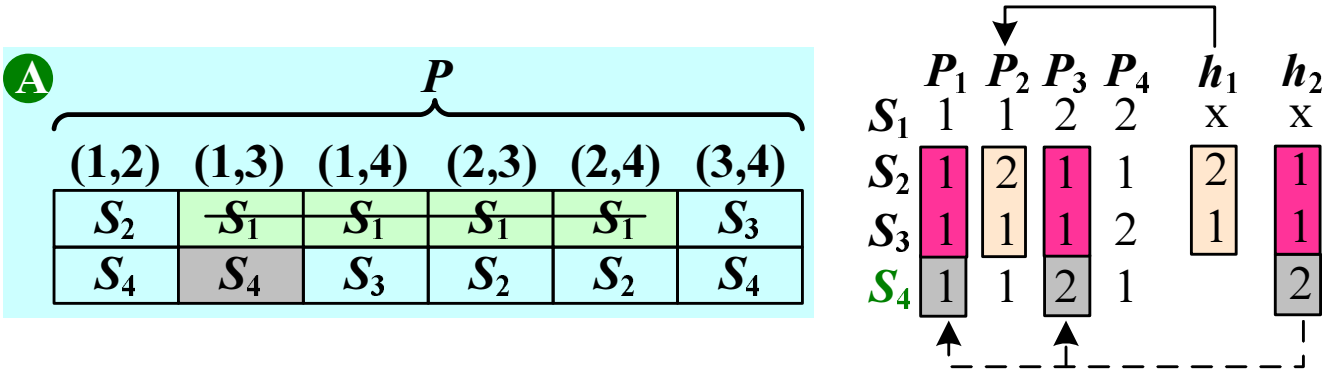

Figure 11
An example of finding auxiliary tag SNPs. The SNP S_1 is missing and SNP S_4 is the auxiliary tag SNP for h_2. (A) The table that stores the set of robust tag SNPs.

0) with probability γ_k (or 1 - γ_k). Let $X_{i,j}$ be the sum of random variables in one inequality of (5), i.e.,

$$X_{i,j} = \sum_{k \in D\{P_i,P_j\}} x_k.$$

By (6), the expected value of $X_{i,j}$ (after randomized rounding) is

$$\begin{aligned} E[X_{i,j}] &= \sum_{k \in D\{P_i,P_j\}} E[x_k] = \sum_{k \in D\{P_i,P_j\}} \gamma_k \\ &\geq m+1. \end{aligned} \quad (7)$$

Lemma 5. *The probability that an inequality in* (5) *is not satisfied after randomized rounding is less than* $e^{-\frac{1}{2(m+1)}}$.

Proof. The probability that an inequality in (5) is not satisfied is $P[X_{i,j} < m+1] = P[X_{i,j} \leq m]$. By the *Chernoff* bound (i.e., $P[X \leq (1-\theta)E[X]] \leq e^{-\frac{\theta^2 E[X]}{2}}$), we have

$$P[X_{i,j} \leq m] \leq e^{-\frac{(E[X_{i,j}]-m)^2}{2E[X_{i,j}]}}. \quad (8)$$

By (7), we know $E[X_{i,j}] \leq m+1$. Since the right-hand side of (8) decreases when $E[X_{i,j}] > m$, we can replace $E[X_{i,j}]$ with $(m+1)$ to obtain an upper bound, i.e.,

$$\begin{aligned} P[X_{i,j} \leq m] &\leq e^{-\frac{(E[X_{i,j}]-m)^2}{2E[X_{i,j}]}} \leq e^{-\frac{(m+1-m)^2}{2(m+1)}} \\ &\leq e^{-\frac{1}{2(m+1)}}. \end{aligned}$$

Theorem 4. *The iterative LP-relaxation algorithm gives a solution of* $O(m \ln K)$ *approximation.*

Proof. Suppose this algorithm runs for t iterations. The probability that all $\frac{K(K-1)}{2}$ inequalities in (5) are satisfied after t iterations is

$$\begin{aligned} \left(1-\left(e^{-1/2(m+1)}\right)^t\right)^{\frac{K(K-1)}{2}} &= \left(1-e^{-t/2(m+1)}\right)^{\frac{K(K-1)}{2}} \\ &\approx e^{-\frac{K(K-1)}{2}e^{-t/2(m+1)}}. \end{aligned}$$

When $t = 2(m+1)\ln\frac{K(K-1)}{2}$, the algorithm stops and returns a solution with probability e^{-1}. Define $OPT(IP)$ and $OPT(LP)$ as the optimal solutions of the IP problem and the LP problem, respectively. Since the solution space of LP includes that of IP,

$$OPT(LP) \leq OPT(IP).$$

Let the set of solutions returned in t iterations be $\{Z_1, Z_2, \ldots, Z_t\}$.

$$E[Z_1] = E[\sum_{k=1}^{N} x_k] = \sum_{k=1}^{N} \gamma_k = OPT(LP).$$

Note that we repeat this algorithm only for those unsatisfied inequalities. Thus, $E[Z_1] \geq E[Z_2] \geq ... \geq E[Z_t]$. Let x_p denote the final solution obtained in Step 4. The expected final solution is

$$\begin{aligned} E[\sum_{p=1}^{N} x_p] &\leq E[\sum_{p=1}^{t} Z_p] \\ &\leq t \times E[Z_1] \\ &\leq t \times OPT(LP) \\ &\leq 2(m+1) \ln \frac{K(K-1)}{2} \times OPT(IP) \\ &= O(m \ln K) \times OPT(IP). \end{aligned}$$

With a high probability, the iterative LP-relaxation algorithm stops after $O(m \ln K)$ iterations and finds a solution of $O(m \ln K)$ approximation.

An algorithm for finding auxiliary tag SNPs

This section describes an algorithm for finding auxiliary tag SNPs assuming robust tag SNPs have been computed in advance. Given a haplotype block M_h containing N SNPs and K haplotypes, we define $C_{tag} \subseteq C$ as the set of tag SNPs genotyped from a haplotype sample with some missing data. This haplotype sample may fail to be distinguished because of the ambiguity caused by missing data. We wish to find the minimum number of auxiliary tag SNPs from the remaining SNPs in the block to resolve the ambiguity. A formal definition of this problem is given below.

Problem: Minimum Auxiliary Tag SNPs (MATS)

Input: An N × K matrix M_h, and a set of SNPs C_{tag} genotyped from a sample with missing data.

Output: The minimum subset of SNPs $C_{aux} \subseteq C - C_{tag}$ such that each pair of ambiguous patterns can be distinguished by SNPs in C_{aux}.

The following theorem shows the NP-hardness of the MATS problem.

Theorem 5. *The MATS problem is NP-hard.*

Proof. Consider that all SNPs in C_{tag} are missing. This special case of the MATS problem becomes finding the minimum tag SNPs from $C - C_{tag}$, which is already known to be NP-hard [17]. Therefore, MATS is also NP-hard.

Although the MATS problem is NP-hard, we show that auxiliary tag SNPs can be found efficiently when robust tag SNPs have been computed in advance. Without loss of generality, assume that these robust tag SNPs are stored in an $(m + 1) \times |P|$ table T_r (see Figure 11 (A)).

Step 1. The patterns that match the haplotype sample are stored into a set A. For example (see Figure 11), if we genotype SNPs S_1, S_2, and S_3 for the sample h_2 and the SNP S_1 is missing, patterns P_1 and P_3 both match h_2. Thus, $A = \{P_1, P_3\}$

Step 2. If $|A| = 1$, the sample is identified unambiguously and we are done (e.g., h_1 in Figure 11). If $|A| > 1$ (e.g., h_2), for each pair of ambiguous patterns in A (e.g., P_1 and P_3), traverse the corresponding column in T_r, find the next unused SNP (e.g., S_4), and add the SNP to C_{aux}. As a result, the SNPs in C_{aux} can distinguish each pair of ambiguous patterns, which are the auxiliary tag SNPs for the haplotype sample.

The worst case of this algorithm is that all SNPs in C_{tag} are missing data, and we need to traverse each column in T_r. Thus, the running time of this algorithm is $O(|T_r|) = O(m|P|)$.

Authors' contributions

YTH and KMC design and implement the greedy algorithms. KZ and TC design and implement the iterative LP relaxation algorithm. All authors write and approve the manuscript.

Additional material

Additional File 1

The program for the first greedy algorithm. *The Greedyl.zip file is compressed using WinZip and contains the JAVA source code for the first greedy algorithm.*
Click here for file
[http://www.biomedcentral.com/content/supplementary/1471-2105-6-263-S1.zip]

Additional File 2

The program for the second greedy algorithm. *The Greedy2.zip file is compressed using WinZip and contains the JAVA source code for the second greedy algorithm.*
Click here for file
[http://www.biomedcentral.com/content/supplementary/1471-2105-6-263-S2.zip]

Additional File 3

The program for the iterative LP-relaxation algorithm. *The ILP.zip file is compressed using WinZip and contains the Perl script for the iterative LP-relaxation algorithm.*
Click here for file
[http://www.biomedcentral.com/content/supplementary/1471-2105-6-263-S3.zip]

Acknowledgements

We thank the referees for their valuable comments that resulted in numerous improvements in the presentation. Yao-Ting Huang and Kun-Mao Chao were supported in part by NSC grants 93-2213-E-002-029 and 94-2213-E-002-091 from the National Science Council, Taiwan. Ting Chen was supported in part by NIH CEGS: Implications of Haplotype Structure in the Human Genome, Grant No. P50 HG002790.

References

1. Bafna V, Halldórsson BV, Schwartz R, Clark AG, Istrail S: Haplotypes and informative SNP selection algorithms: don't block out information. *Proc RECOMB'03* 2003:19-27.
2. Carlson CS, Eberle MA, Rieder MJ, Yi Q, Kruglyak L, Nickerson DA: **Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium.** *Am J Hum Genet* 2004, **74:**106-120.
3. Cormen TH, Leiserson CE, Rivest RL, Stein C: *Introduction to algorithms* The MIT Press; 2001.
4. Daly MJ, Rioux JD, Schaffner SF, Hudson TJ, Lander ES: **High-resolution haplotype structure in the human genome.** *Nat Genet* 2001, **29(2):**229-232.
5. Forsgren A, Gill PE, Wright MH: **Interior methods for nonlinear optimization.** *SIAM Rev* 2002, **44:**525-597.
6. Garey MR, Johnson DS: *Computers and intractability* Freeman, New York; 1979.
7. Halldórsson BV, Bafna V, Lippert R, Schwartz R, Vega FM, Clark AG, Istrail S: **Optimal haplotype block-free selection of tagging SNPs for genome-wide association studies.** *Genome Research* 2004:1633-1640.
8. Halperin E, Eskin E: **Haplotype reconstruction from genotype data using imperfect phylogeny.** *Bioinformatics* 2004.
9. Hinds DA, Stuve LL, Nilsen GB, Halperin E, Eskin E, Ballinger DG, Frazer KA, Cox DR: **Whole-genome patterns of common DNA variation in three human populations.** *Science* 2005, **307:**1072-1079.
10. Hudson RR: **Generating samples under a Wright-Fisher neutral model of genetic variation.** *Bioinformatics* 2002, **18:**337-338.
11. **LP Solve** [http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml]
12. Niu T, Qin Z, Xu X, Liu JS: **Bayesian haplotype inference for multiple linked single-nucleotide polymorphisms.** *Am J Hum Genet* 2002, **70:**157-159.
13. Patil N, Berno AJ, Hinds DA, Barrett WA, Doshi JM, Hacker CR, Kautzer CR, Lee DH, Marjoribanks C, McDonough DP, Nguyen BT, Norris MC, Sheehan JB, Shen N, Stern D, Stokowski RP, Thomas DJ, Trulson MO, Vyas KR, Frazer KA, Fodor SP, Cox DR: **Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21.** *Science* 2001, **294:**1719-1723.
14. Stephens M, Donnelly P: **A comparison of bayesian methods for haplotype reconstruction from population genotype data.** *Am J Hum Genet* 2003, **73:**1162-1169.
15. Wang L, Xu Y: **Haplotype inference by maximum parsimony.** *Bioinformatics* 2003, **19(14):**1773-1780.
16. Yang Y, Zhang J, Hoh J, Matsuda F, Xu P, Lathrop M, Ott J: **Efficiency of single-nucleotide polymorphism haplotype estimation from pooled DNA.** *Proc Nat Acad Set* 2003, **100(12):**7225-7230.
17. Zhang K, Deng M, Chen T, Waterman MS, Sun F: **A dynamic programming algorithm for haplotype partitioning.** *Proc Nat Acad Sci* 2002, **99(11):**7335-7339.
18. Zhang K, Sun F, Waterman MS, Chen T: **Haplotype block partition with limited resources and applications to human chromosome 21 haplotype data.** *Am J Hum Genet* 2003, **73:**63-73.
19. Zhang K, Qin ZS, Liu JS, Chen T, Waterman MS, Sun F: **Haplotype block partition and tag SNP selection using genotype data and their applications to association studies.** *Genome Research* 2004, **14:**908-916.
20. Zhao JH, Lissarrague S, Essioux L, Sham PC: **GENECOUNTING: haplotype analysis with missing genotypes.** *Bioinformatics* 2002, **18:**1694-1695.